数式処理システムGALのパーザ(数式処理と数学研究への応用)

AUTHOR(S):

佐々木, 建昭

CITATION:

佐々木, 建昭. 数式処理システムGALのパーザ(数式処理と数学研究への応用). 数理解析研究所講究録 1987, 612: 101-110

ISSUE DATE:

1987-02

URL:

http://hdl.handle.net/2433/99775

数理解析研究所講究録
612巻 1987年 101-110

数式処理システムGALのパーザ

理化学研究所，佐々木建昭（Tateaki Sasaki）

1. はじめに

国産数式処理システムGALでは，システムプログラミング用とユーザ用の言語のために独自のパーザを開発したが，それは以下の理由による：

① 広汎な数式処理のためには記号・言語体系を設計し直す必要があり，従来の数式処理用パーザでは不十分である。

② 大規模システムではパッケージ化（モジュール化）が不可欠であるが，それをGALのレベルでサポートする。

③ 外国製のパーザを使うことは著作権侵害である。

現在ではパーザ開発は俗事で，YACCというパーザ開発用ツールもあるほどである。が，筆者はYACCを使う術も知らず，他のパーザや書物を詳しく調べることもなく，全く我流で開発した。そのおかげで，パーザの中味は全く独自のものであるし，またスミからスミまで理解しているから，機能の追加修正に際しては不便をほとんど感じない。機能の追加・修正は今後何度にもわたって行なわれるであろうことを考えると，

国産システムの独自開発の重要性を強く感じる。

本稿では，上記①～③を中心に，GALのパーザの機能を紹介する。なお，REDUCEのRLISPと同様，GALのパーザはユーザ用言語とSyntax-Sugared Lisp（略してSSLisp）の両方のパージングを受け持つ。このモードの切り替えはENTERコマンド（たとえば“ENTER GAL;”のように使用される）でなされる。

2．GALにおける記号体系

GAL計画では，最終的にはREDUCEなどよりもはるかに広範囲の数式を扱う予定であり，内部表現がそのように設計されている。広範囲の数式を限定されたキーボードから入力することを考えると，特殊文字をきびしく節約して使わざるを得ない。（もちろん，数学的記号を多く含む端末をメーカーが提供してくれるとよいのだが，期待薄であろう）。この観点から見ると，REDUCEやMACSYMAにおける特殊文字の使い方はぜいたくである。たとえば，終端記号に二つの特殊文字を使っている（出力促進用の；と出力抑制用の$）．因みに，GALでは終端記号は出力促進用が；，出力抑制用が;; となっている。

GALパーザはユーザ用言語とSSLispの両方を受け持つか

ら，もしも記号体系を両者に共通にすれば，それは非常に窮屈なものとなろう。幸いなことに，Lispと数学では記号の用法が大きく異なるから，GALではLispと数学用に2種類の記号体系を用いている。したがって，多くの特殊文字に複数の意味が与えられている。（この結果として，Lispモードと GALモードを頻繁に切り替えると，筆者でさえ用法を混同することがある。これを避けるため，GALでは，GALモードのユーザはLispモードに移ることなく必要な仕事ができるよう，可能な限り工夫する積りである）。

以下，現在インプリメントされている特殊文字の用法のいくつかを示す。なお，特殊文字から成るトークンは，前置型，binary内置型，nary内置型，置換内置型（他のトークンで置き換えられる），後置型の5種があり，

DEFTOKEN（MODE，CHS，TYPE，NAME，AFTER）

MODE：'LISP あるいは 'GAL，

CHS：トークンを構成する文字のリスト，

TYPE：上記5種の型のどれか，

NAME：トークンの名前（ほとんどの場合，対応するプロシジャーの名前である），

AFTER：トークンの順位を指定（どのトークンの後にするかを指定），

により，いくらでも新しいトークンを定義できる。

トークン	GALでの意味	SSLisp中
≪・・・≫	グループ文	同じ
(・・・)	通常のかっこ	同じ
[・・・]	添字用（多重も可）	———
{・・・}	集合を表わす	———
⟨{・・・}⟩	順序集合を表わす	———
{・・・|・・・}	条件つき集合	———
||	コメント記号	同じ
!	階乗記号（後置トークン）	エスケープ記号
!!	2重階乗記号（〃）	———
'	評価抑制（前置トークン）	QUOTE
"・・・"	文字列（WRITE文中で）	同じ
=	等式関係	EQUAL
?=	質問関数（等しいか？）	EQUAL
==>	左→右への書き換え定義	———
%	前式の引用（%(・・・)も可）	———
%・・・	パッケージ内名前に使用	同じ
@・・・	パターン変数を表わす	———
#・・・	超越数／代数的数	———

3. 言語体系

現在わが国では，数式処理の代名詞＝REDUCEというくらいにREDUCEがよく普及しており，ユーザはREDUCEの言語体系によく慣れ親しんでいる。ユーザのREDUCE言語に対する親和性を最大限尊重し，GALの初期設計の段階では「GALの言語体系はREDUCEのそれと上位互換にする」積りで，本研究会でもそのように発言してきた。しかし，出来あがりはそうはならなかった。前言を撤回するとともに，その言い訳をしておかねばなるまい。

たとえば記号体系について言えば，前節に見たように，広汎な数式を扱うには記号体系を再検討する必要があったこと。REDUCEのLET文は誤解を招き易く，また多項書き換え則への拡張が強く望まれていたこと（GALではこれはRULE文で実現された）。GALでは多数の型の数式がサポートされ，それらの処理のために新たな宣言文が必要になったこと（宣言文はDECLARE文であるが，これにより代数的数なども定義できる）。さらに数学公式データベース用言語も装備する必要があること（データベースとその検索言語の骨子は現在ほぼ固まった段階にある）。これらの事情を考慮すると，GALの言語体系をREDUCEのそれと上位互換にすることは無理であると，理解して頂けるものと思う。

なお，REDUCEの言語体系のうち，残せるものはできる限り残した。その最たるものはRLISP ≈ SSLispをシステム記述に用いることである。これは，作成者の立場から言えばやりたくないことである（何故なら，人の物真似と言われるにきまっているから）。この不満は残るものの，GAL パーザは全く独自に開発したから，著作権の点では外国に対しても胸が張れると思う。

4. 新しい文の定義

ユーザ言語あるいはシステム記述言語において，“IF”，“THEN”，“FOR”，“DO” などのように，キーとなる語をキーワードと呼ぶ。文はこれらのキーワードを用いて構成されるから，豊富な機能を備えるためには，キーワードの個数が増えるのは避けられない。一方，ユーザにとっては，キーワードは少ないほど楽である（たとえばREDUCEではSUMはキーワードだが，プロシジャ変数としてSUMを用いてエラーを引き起こさなかったユーザは少数派であろう）。さらに，キーワードが多いことは語句解析の効率を落とすことにもつながる。何故なら，入力文中のあらゆる名前は，まず最初にキーワードかどうか判定されるのであるから（この判定は非常に高速になされるので，効率の低下は実際上は無視で

きるほどである）。

さて、GALでは、“DECLARE”や“FIND”等に多数の機能を負わせてキーワードの個数の増加を押さえたのみならず、GALとSSLispでキーワードを別建てとした。これにより、GALのキーワードはLispのプログラミングには何ら影響を及ぼさない。

新しいキーワードはDEFKEYWORDで定義できる：

DEFKEYWORD(MODE, WORD, PROCname)

ただし、MODEは'GAL, 'LISP, 'BOTHのいずれかであり、WORDはキーワード名、PROCnameはそのキーワードの処理ルーチン名である。ただし、PROCnameが'PAUSEのときはWORDは区切り語となる（たとえば“THEN”や“ELSE”がそうである）。

ユーザが新しい文を導入する際の問題点は、「キーワード処理ルーチンをいかに書くか？」である。今のところ、これを自動化することは全く考えておらず、各自が書かなければならないが、それは比較的容易であると考える。なぜなら、GALのパーザは二つのプロシジャがほとんどの仕事をするが、そのうちの一つ、READ.STATの仕様を理解しさえすれば処理ルーチンが書けるからである。

5. パッケージ化機能

パッケージ化の意味は次のものとする：

「パッケージAとBで，パッケージ内記号として同名のものを用いても，AとBでは別物として扱われる」

大規模システムで，別人の手になる多数のモジュールから構成されるシステムにおいては，名前の衝突を避けるために，パッケージ化の機能が不可欠である。

GALのパーザでは3種のパッケージ化がなされる：

① ホストであるLisp処理系に対してGALシステム全体が一つのパッケージとなる（この機能がなければ，Lispの大域変数は危険で使えなくなるであろう）；

② プロシジャの引数とBEGIN文のLOCAL変数は，プロシジャ名等と衝突しないように改名される；

③ Lispシステムと GALシステムの各々の内部で，一群のプロシジャをパッケージ化できる。

パッケージ化の方法は，①ではGAL用記号名には頭にブランクをつけ，②ではLOCAL記号名の頭にコンマをつけ，③ではパッケージPP内の%nameをPP␣name（GAL用記号に対して）あるいはPP,name（Lisp用記号に対して）に改名するだけである。しかって，パッケージ化は完全とは言えないが，名前の衝突が大巾に減少することは確実である。（パ

ッケージ化を完全に実現するには，Common Lisp のように各パッケージに対して別々の記号表を用いればよいが，これをGALのパーザレベルでサポートするのは荷が重すぎる。）

簡単な入力プログラムの翻訳例を示し，Lispモードと GALモードで翻訳がどう異なるかを説明しよう。

```
PROCEDURE FOO(A,B) ==
BEGIN LOCAL U,V;
      U := SIN(X) + A;
      V := COS(Y) + B;
      RETURN U+V;  END;;
```

入力プログラム例

上記の入力はLispモードでは次のように翻訳され，

```
(DE FOO (!,A !,B)
   (PROG (!,U !,V)
      (SETQ !,U (PLUS (SIN X) !,A))
      (SETQ !,V (PLUS (COS Y) !,B))
      (RETURN (PLUS !,U !,V))))
```

GALモードでは次のように翻訳される。

```
(DE ! FOO (!,A !,B)
   (PROG (!,U !,V)
(SETQ !,U (! PLUS (GEV!:LIST '! SIN (GEV!:ID '! X)) !,A))
(SETQ !,V (! PLUS (GEV!:LIST '! COS (GEV!:ID '! Y)) '!,B))
(RETURN (! PLUS !,U !,V))))
```

見て分る通り，翻訳結果は大いに異なる。上例においては，局所変数と関数名の改名に加えて，GAL用プログラムにおける 'GEV!:ID と 'GEV!:LIST に注目されたい。GEV!:IDはその引数をGALの記号として処理し（GAL用の記号表に登録

し，型と順位数を与え，必要ならば多項式の内部表現に変換して返す），GEV!:LIST はリスト形式のGAL入力を処理する（たとえば関数の内部表現を構成して返す）。上記のGAL用プログラムはコンパイルすれば高速に処理しうることに注目されたい。

6．今後の課題

現在，入力ミスを修正するエディタを開発中であるが，GALのパーザはエディティングに便利なように構成してあり，開発は短期間で終了すると思われる。

より重要なことは，GALの中間層の言語（アルゴリズム・インプリメンテイション用）の整備である。GALでは，アルゴリズムのインプリが，Lispで行なうようにきめ細かく，かつ（Lispの知識がなくても）数学的知識だけで容易に実行できるようにしたいのである。